МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РФ

Федеральное государственное автономное
образовательное учреждение высшего образования
«Национальный исследовательский университет ИТМО»

ФАКУЛЬТЕТ ПРОГРАММНОЙ ИНЖЕНЕРИИ И КОМПЬЮТЕРНОЙ ТЕХНИКИ

ЛАБОРАТОРНАЯ РАБОТА №5

по дисциплине

‘Системы искусственного интеллекта‘

Выполнил:
Студент группы P33312
Соболев Иван
Александрович
Преподаватель:
Кугаевских Александр
Владимирович

УНИВЕРСИТЕТ ИТМО

Санкт-Петербург, 2023

Задание:

Этапы реализации и пояснения:

На первых шагах были импортированы нужные библиотеки, далее с помощью библиотеки pandas считан датасет.

Дальше необходимо было сделать предварительную обработку и нормировку данных, для этого была написана функция нормирования.

```
# Масштабирование
def MinMaxScaler(A):
    for i in A.columns:
        maxi = A[i].max()
        mini = A[i].min()
        A[i] = (A[i] - mini) / (maxi - mini)
    return A

# Разделение на обучающий и тестовый наборы
def train_test_split_custom(X, y, test_size=0.2):
    num_samples = X.shape[0]
    num_test_samples = int(test_size * num_samples)

    # Генерация случайных индексов для тестового набора
    test_indices = np.random.choice(num_samples, num_test_samples, replace=False)

    # Индексы для обучающего набора
    train_indices = np.setdiff1d(np.arange(num_samples), test_indices)

    X_train, X_test = X.iloc[train_indices], X.iloc[test_indices]
    y_train, y_test = y.iloc[train_indices], y.iloc[test_indices]

    return X_train, X_test, y_train, y_test
```

Создаем функцию train_test_split_custom, которая разделяет данные на обучающий и тестовый наборы. Эта функция случайным образом выбирает индексы для тестового набора данных, исходя из заданного коэффициента test_size. Таким образом, мы получаем два набора данных: X_train, y_train - обучающий набор, и X_test, y_test - тестовый набор.

Дальше создаём основной модуль.

```
class KNN:
    def __init__(self, k=3):
        self.k = k
        self.X_train = None
        self.y_train = None

    def fit(self, X, y):
        self.X_train = X
        self.y_train = y

    # Евклидово расстояние
    def distance(self, x0, x1):
        return np.sqrt(np.sum((x0 - x1)**2))

    # Наиболее частый класс
    def most_common(self, y):
        labels = np.unique(y)
        count = [list(y).count(i) for i in labels]
        return labels[np.argmax(count)]

    def predict(self, X_test):
        # Предсказываем метки классов
        labels = [self.find_labels(x) for x in X_test]
        return np.array(labels)

    def find_labels(self, x):
        # Считаем расстояние
        distances = [self.distance(x, x_train) for x_train in self.X_train]
        # Берем индексы наблюдений
        k_nearest = np.argsort(distances)[:self.k]
        # По индексам берем метки классов
        labels = [self.y_train[i] for i in k_nearest]
        return self.most_common(labels)
```

Далее просто создаем несколько моделей и анализируем их.

```
# Модель со случайными признаками
tags = ["Alcohol", "Malic Acid","Ash","Alcalinity of ash","Magnes
n = random.randint(1,13)
tags_1 = random.sample(tags, n)
print(tags_1)

X_test_rand = X_test[tags_1]
X_train_rand = X_train[tags_1]
X_test_rand=X_test_rand.to_numpy()
X_train_rand = X_train_rand.to_numpy()
k=[]
test_score = []
for i in range(3,21,1):
  clf = KNN(k=i)
  clf.fit(X_train_rand,y_train_np)
  y_pred = clf.predict(X_test_rand)
  show_cf_matrix(confusion_matrix(y_test_np, y_pred))
  test_score.append(f1_score(y_test_np,y_pred))
  k.append(i)

plt.plot(k,test_score)
plt.show
```

```
# Модель с фиксированными признаками
k=[]
test_score = []
tags = ["Total phenols","Alcohol","Color intensity"]
X_test_fix = X_test[tags]
X_train_fix = X_train[tags]
X_test_fix=X_test_fix.to_numpy()
X_train_fix = X_train_fix.to_numpy()
k=[]
test_score = []
for i in range(3,21,1):
  clf = KNN(k=i)
  clf.fit(X_train_fix,y_train_np)
  y_pred = clf.predict(X_test_fix)
  show_cf_matrix(confusion_matrix(y_test_np, y_pred))
  test_score.append(f1_score(y_test_np,y_pred))
  k.append(i)

plt.plot(k,test_score)
plt.show
```

Выводы:

Можно заметить, что с увеличением количества ближайших соседей показатель f1_score падает. Оптимальным количеством соседей является 5-7 для данного набора тренировочных и тестовых данных.